CENG 3420 Computer Organization & Design

Lecture 07: Arithmetic and Logic Unit – 2

Bei Yu
CSE Department, CUHK
`byu@cse.cuhk.edu.hk`

(Latest update: January 20, 2022)

Spring 2022

Multiplication & Division

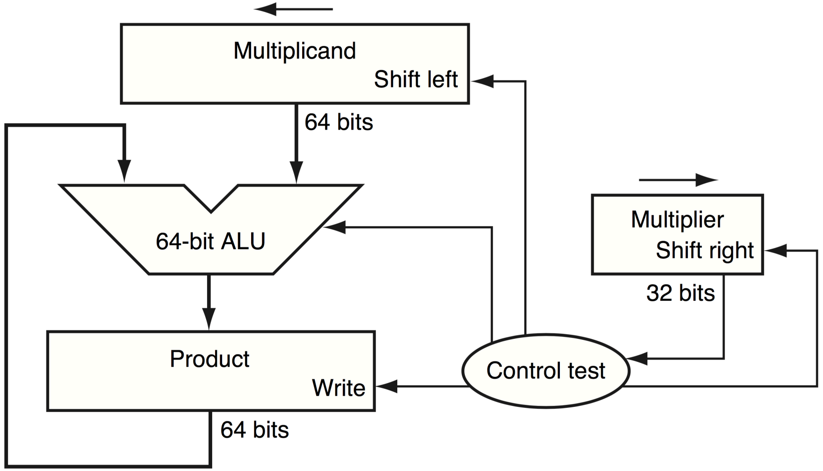
Multiplication

- More complicated than addition
- Can be accomplished via shifting and adding

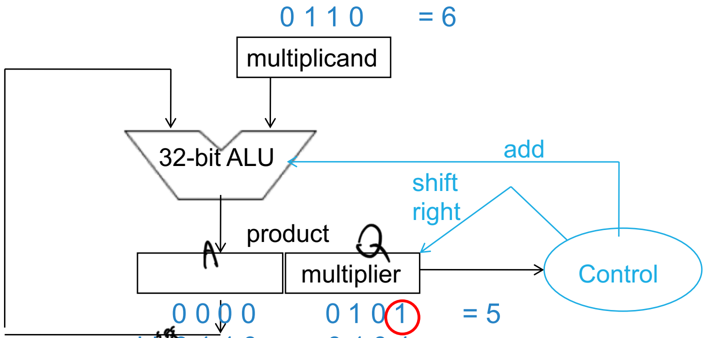
$$\begin{array}{rl}
0010 & \text{(multiplicand)}\\
\times\ 1011 & \text{(multiplier)}\\
\hline
\left.\begin{array}{r} 0010\\ 0010\\ 0000\\ 0010\end{array}\right\} & \text{(partial product array)}\\
\hline
\boxed{0001}\boxed{0110} & \text{(product)}
\end{array}$$

- Double precision product produced
- More time and more area to compute

First Version of Multiplication Hardware



Add and Right Shift Multiplier Hardware



```
      0 0 0 0        0 1 0 1    = 5
add   0 1 1 0        0 1 0 1
shift 0 0 1 1  →     0 0 1 0
add   0 0 1 1        0 0 1 0
shift 0 0 0 1  →     1 0 0 1
add   0 1 1 1        1 0 0 1
shift 0 0 1 1  →     1 1 0 0
add   0 0 1 1        1 1 0 0
shift 0 0 0 1  →     1 1 1 0    = 30
            16       8 4 2
```

bit 有几多,
shift 几多

RISC-V Multiply Instruction

- Multiply (`mult` and `multu`) produces a double precision product

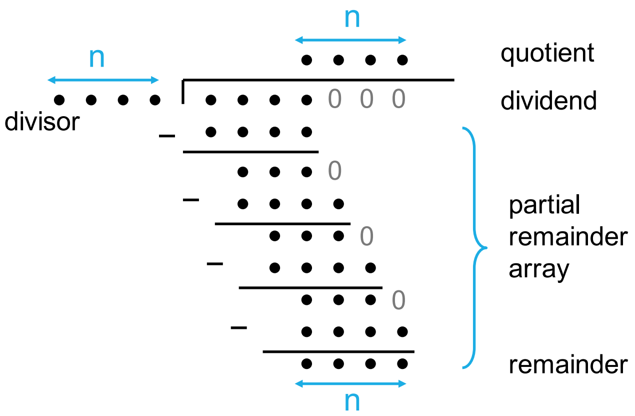
```
mul $rd, $s0, $s1        # hi||lo = $s0 * $s1
```

0	16	17	0	0	0x18

- Low-order word of the product is left in processor register `lo` and the high-order word is left in register `hi`
- Instructions `mfhi rd` and `mflo rd` are provided to move the product to (user accessible) registers in the register file
- Multiplies are usually done by fast, dedicated hardware and are much more complex (and slower) than adders

Division

- Division is just a bunch of quotient digit guesses and left shifts and subtracts



Question: Division

Dividing 1001010 by 1000

```
            1001
        _________
1000 )  1001010
        1000
        ____
          0010
          0000
          ____
            0101
            1000
            ____
             1010
             1000
             ____
               10
```

RISC-V Divide Instruction

- Divide generates the reminder in `hi` and the quotient in `lo`

```
div $rd, $s0, $s1        # lo = $s0 / $s1
                         # hi = $s0 mod $s1
```

op	rs	rt	rd	shamt	funct

- Instructions `mflo rd` and `mfhi rd` are provided to move the quotient and reminder to (user accessible) registers in the register file
- As with multiply, divide ignores overflow so software must determine if the quotient is too large.
- Software must also check the divisor to avoid division by 0.

Shift

Shift Operations

- Shifts move all the bits in a word left or right

```
sll  $t2, $s0, 8 #$t2 = $s0 << 8 bits
srl  $t2, $s0, 8 #$t2 = $s0 >> 8 bits
sra  $t2, $s0, 8 #$t2 = $s0 >> 8 bits
```

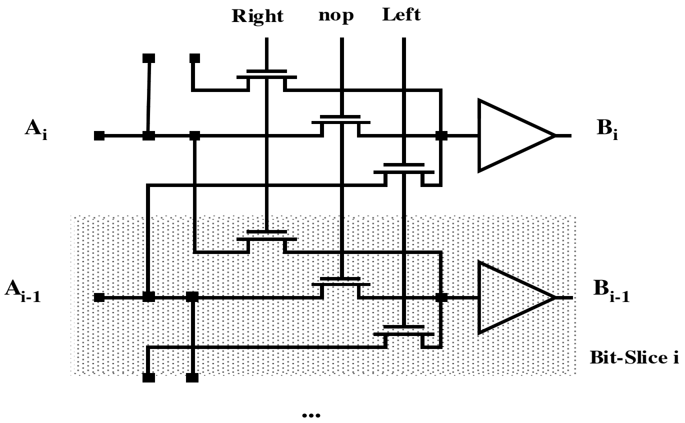
op	rs	rt	rd	shamt	funct

- Notice that a 5-bit `shamt` field is enough to shift a 32-bit value $2^5 - 1$ or 31 bit positions
- Logical shifts fill with zeros, arithmetic left shifts fill with the sign bit

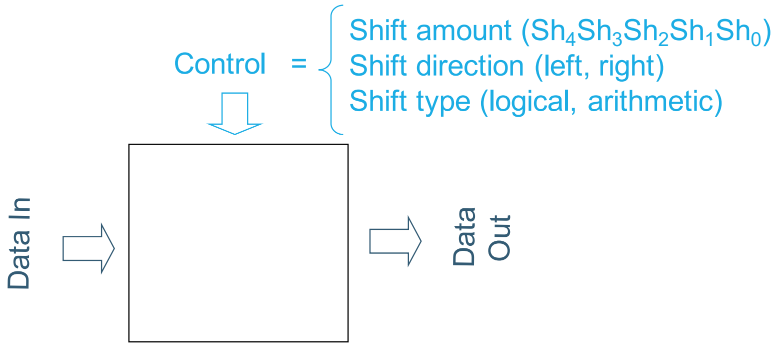
The shift operation is implemented by hardware separate from the ALU

Using a barrel shifter, which would takes lots of gates in discrete logic, but is pretty easy to implement in VLSI

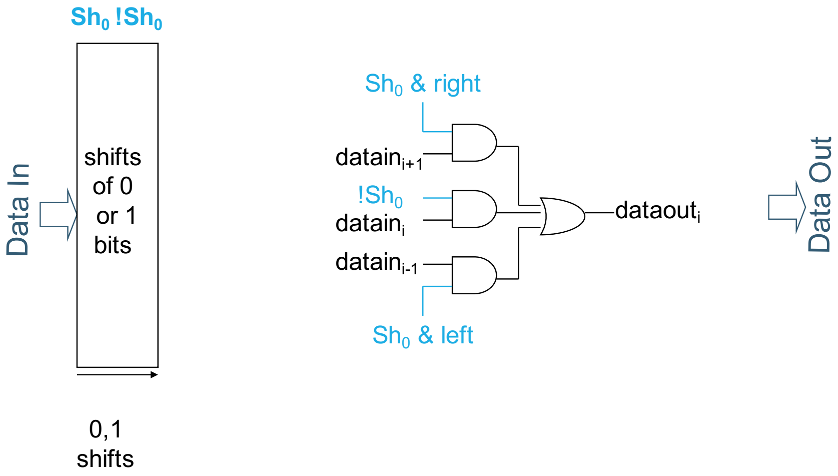
A Simple Shifter



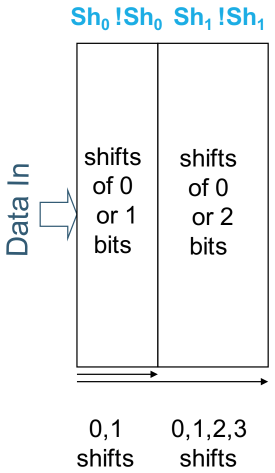
Parallel Programmable Shifters

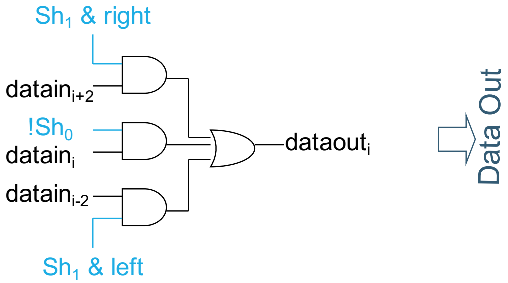

Control =
Shift amount ($Sh_4Sh_3Sh_2Sh_1Sh_0$)
Shift direction (left, right)
Shift type (logical, arithmetic)
Data In
Data Out

Logarithmic Shifter Structure

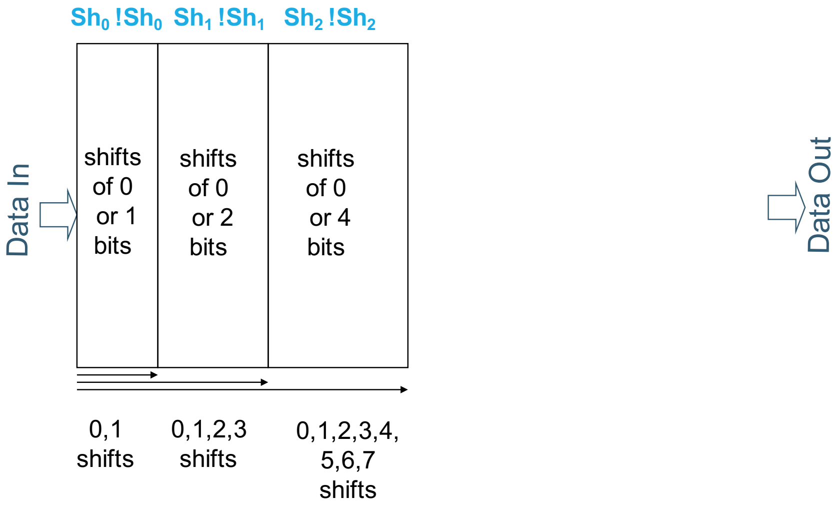


Logarithmic Shifter Structure

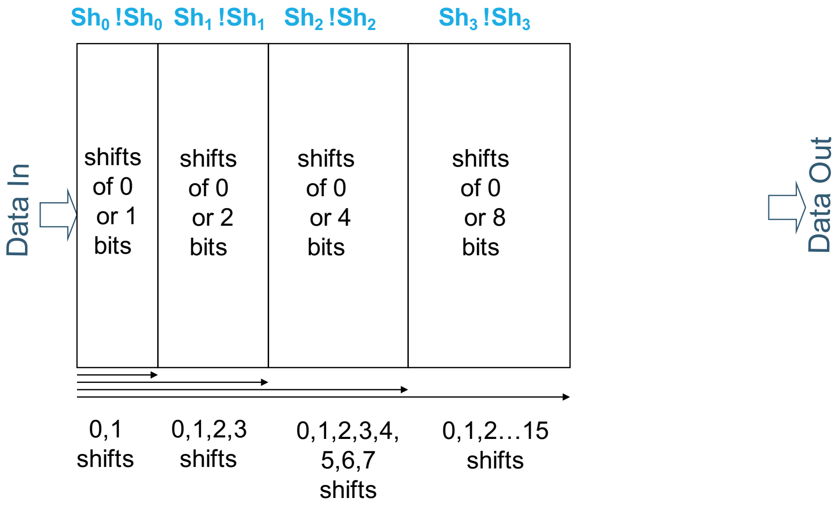




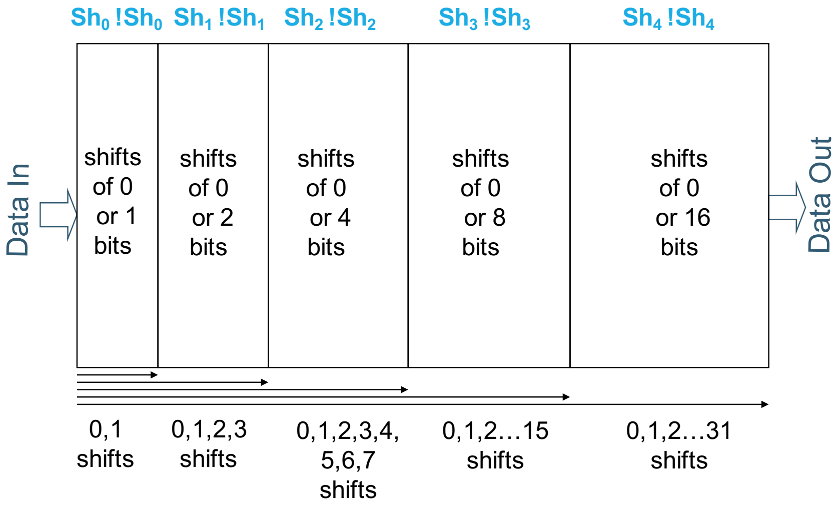
Logarithmic Shifter Structure



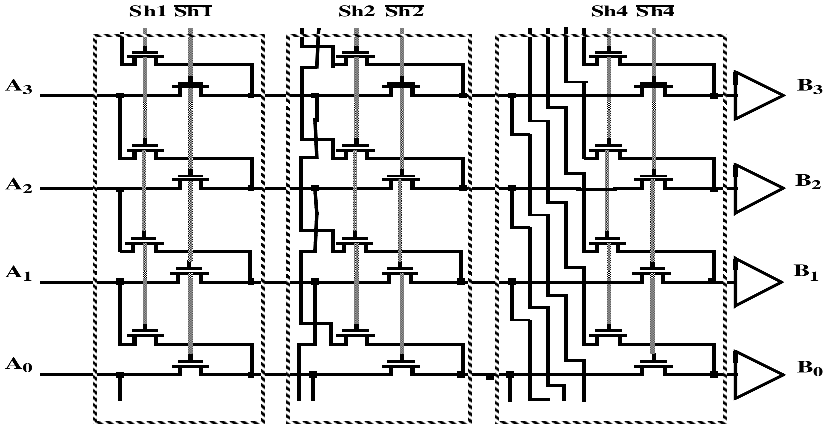
Logarithmic Shifter Structure



Logarithmic Shifter Structure



Logarithmic Shifter Structure



Sh1 $\overline{\text{Sh1}}$
Sh2 $\overline{\text{Sh2}}$
Sh4 $\overline{\text{Sh4}}$
A_3
A_2
A_1
A_0
B_3
B_2
B_1
B_0